Programación Python para BigData

Lección 4: PostgreSQL

eip ESCUELA INTERNACIONAL POSGRADOS

Indice

Introducción

En el tema que nos ocupa, se han explicado varias formas de trabajar con PostgreSQL y en la actividad se pide realizar empleando Adminer, una serie de tareas con una base de datos.

Trabajando con Adminer

Arrancando Adminer

Antes de nada arrancaremos el servicio empleando Docker-Compose, para ello será necesario emplear el archivo yml adjunto a este documento. Para arrancar el servicio escribiremos lo siguiente en la consola de comandos:

```
sudo docker-compose -f <ruta del archivo>/postgres.yml up
```

Únicamente deberemos sustituir <ruta del archivo> por la ruta de la carpeta en la que tengamos guardado el archivo yml esperaremos unos segundos y accederemos en el buscador de Internet a la dirección http://0.0.0.0:8080/.

Accediendo a la base de datos

Una vez hayamos accedido a través de Internet a la dirección http://0.0.0.0:8080/ veremos la siguiente pantalla:

En el campo “Motor de Base de datos” seleccionaremos PostgreSQL y en Usuario y contraseña pondremos los establecidos en el archivo yml, en mi caso Elidas y EIPpython.

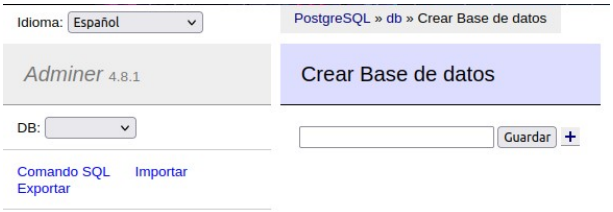
Una vez accedamos, veremos le siguiente pantalla:

Trabajando en Adminer

Creando una base de datos:

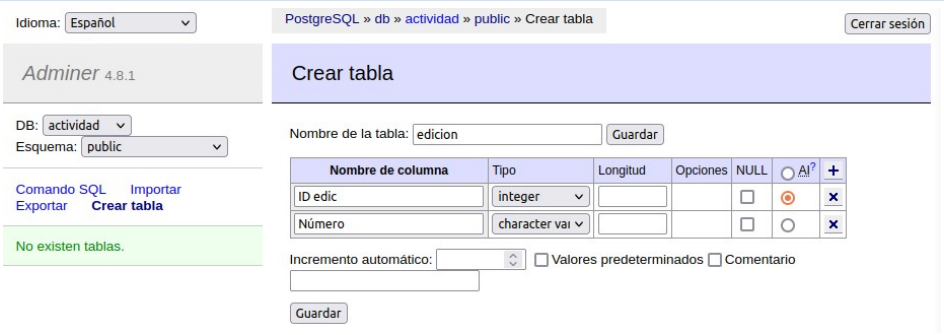
Una vez hayamos accedido con nuestro usuario a Adminer procederemos a crear una base de datos. Para ello, clicaremos en el botón “Crear Base de datos”,

le daremos un nombre, en nuestro caso será “actividad” y le daremos a “Guardar”.



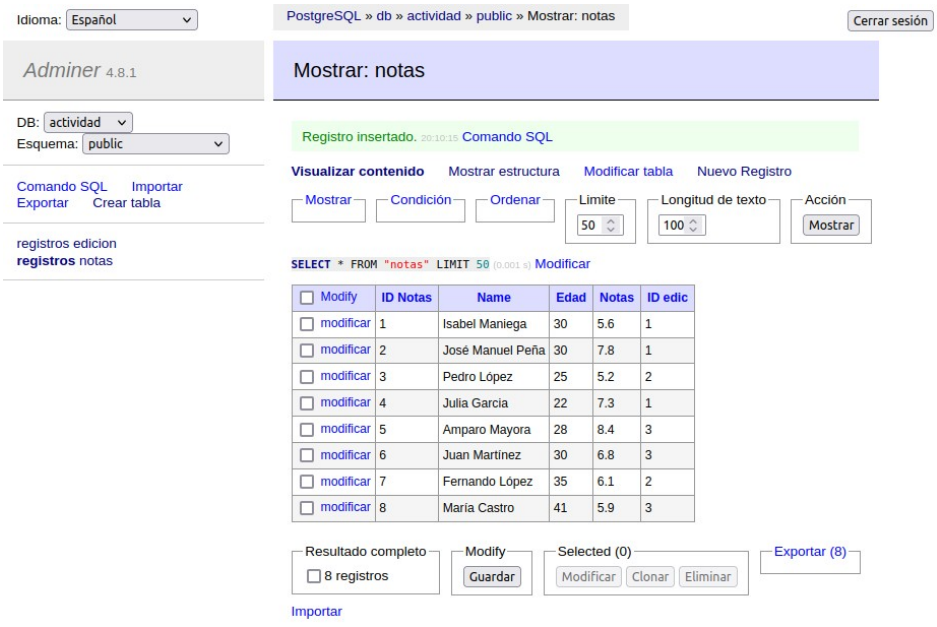
Una vez creada, procederemos a crear las tablas de la base de datos. En este caso crearemos dos, una de nombre “notas” y otra de nombre “edicion” para ello clicaremos en “Crear tabla”.

Lo siguiente será crear las columnas, empezaremos por la tabla “edicion” que deberá tener el siguiente aspecto:



Una vez creadas las columnas, introduciremos el contenido mediante el botón “Nuevo Registro”.

Tras esto, crearemos la base de datos Notas cuyo resultado final ha de ser algo así:



Creando un script

Dado que la solución que se pide es singular, he optado por crear un script de tal manera que en un futuro sea generalizable. Para ello he creado una clase:

__init__(self, name)

```python
# __LIBRARIES__ #
import psycopg2 as psy  # por hacerlo mas comodo
from psycopg2 import sql
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT


# __MAIN CODE__ #
class Postgres:
    def __init__(self, name):
        self.name = name
        self.conect()
        self.edic = ['Uno', 'Dos', 'Tres']
        self.notas = [
            ('Isabel Maniega', 30, 5.6, 1), ('José Manuel Peña', 30, 7.8, 1),
            ('Pedro López', 25, 5.2, 2), ('Julia García', 22, 7.3, 1),
            ('Amparo Mayora', 28, 8.4, 3), ('Juan Martínez', 30, 6.8, 3),
            ('Fernando López', 35, 6.1, 2), ('María Castro', 41, 5.9, 3)
        ]
```

Como se aprecia en la imagen en el __init__ se introduce de antemano la información con la que se trabajará en este caso, ademas se hace una llamada a la función conect(). La única información que, en este caso, tiene que introducir el usuario será el nombre de la base de datos.

conect(self):

Dado que es necesario realizar dos conexiones, he optado por realizar una función recursiva que, en el primer bucle intenta conectarse a la base de datos que se le ha indicado y de no conseguirlo, se conecta al servidor empleando los credenciales que se le han dado y crea la base de datos además de crear el cursor para poder ejecutar funciones mas adelante.

```python
    def conect(self):
        try:
            # conectemonos a la base de datos:
            self.conn = psy.connect(
                database=self.name,
                user='Elidas', password='MaulLilo3125',
                host='localhost', port=5432
            )
            # creamos el cursor
            self.cur = self.conn.cursor()
            return True

        except psy.Error:
            # Creamos la conexión
            conn = psy.connect(
                user='Elidas', password='MaulLilo3125',
                host='localhost', port=5432
            )

            conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)

            # creamos el cursor
            cur = conn.cursor()

            # creamos la base de datos:
            cur.execute(
                sql.SQL('CREATE DATABASE {};').format(
                    sql.Identifier(self.name)
                )
            )  # Nota 1.0 & 1.1

            return self.conect()
```

create_table(self):

Crea las dos tablas requeridas con sus columnas de forma simultánea, en caso de estar creadas, devuelve un error.

```
def create_table(self):  # creamos las dos tablas
    try:
        self.cur.execute(
            "CREATE TABLE edicion( \
                IDedic serial, numero varchar);"
        )
        self.cur.execute(
            "CREATE TABLE notas( \
                IDnotas serial, name varchar(120),\
                edad int, notas real, IDedic int);"
        )

    except psy.Error as e:
        print('Error Crear tabla: %s', str(e))

    self.conn.commit()
```

fill_edic(self) y fill_notas(self):

Ambas funciones obtienen los datos de las listas creadas en el __init__ además, hacen uso de un contador y la recursividad para recorrer estas listas y poder incluir grandes cantidades de información de forma rápida y ahorrando líneas de código.

```
def fill_edic(self, cnt=0):
    while cnt != len(self.edic):
        try:
            self.cur.execute("INSERT INTO edicion VALUES\
                (nextval('edicion_idedic_seq'), %s)",
                            (self.edic[cnt],))

        except psy.Error as e:
            print('Error Insertar dato: %s', str(e))
        cnt += 1
        self.conn.commit()
        return self.fill_edic(cnt)
    return True

def fill_notas(self, cnt=0):
    while cnt != len(self.notas):
        try:
            self.cur.execute("INSERT INTO notas VALUES\
                (nextval('notas_idnotas_seq'), %s, %s, %s, %s)",
                            self.notas[cnt])  # ('Oscar', 25, 5.0))

        except psy.Error as e:
            print('Error Insertar dato: %s', str(e))
        cnt += 1
        self.conn.commit()
        return self.fill_notas(cnt)
    return True
```

El valor intoducido como dato “nextval_notas_idnotas_seq” hace referencia al hecho de que la variable en esa columna es secuencial y por tanto no hay que darle valor alguno.

Una vez llegados a este punto, las bases de datos tienen el siguiente aspecto:

Mostrar: edicion

Visualizar contenido | Mostrar estructura | Modificar tabla | Nuevo Registro

Mostrar | Condición | Ordenar | Limite 50 | Longitud de texto 100 | Acción: Mostrar

`SELECT * FROM "edicion" LIMIT 50` (0.001 s) Modificar

Modify	idedic	numero
modificar	1	Uno
modificar	2	Dos
modificar	3	Tres

Mostrar: notas

Visualizar contenido | Mostrar estructura | Modificar tabla | Nuevo Registro

Mostrar | Condición | Ordenar | Limite 50 | Longitud de texto 100 | Acción: Mostrar

`SELECT * FROM "notas" LIMIT 50` (0.000 s) Modificar

Modify	idnotas	name	edad	notas	idedic
modificar	1	Isabel Maniega	30	5.6	1
modificar	2	José Manuel Peña	30	7.8	1
modificar	3	Pedro López	25	5.2	2
modificar	4	Julia García	22	7.3	1
modificar	5	Amparo Mayora	28	8.4	3
modificar	6	Juan Martínez	30	6.8	3
modificar	7	Fernando López	35	6.1	2
modificar	8	María Castro	41	5.9	3

update(self):

Haciendo uso de la columna “idnotas”, cuyo valor es único, actualiza los valores de la columna seleccionada cuyo id coincida con el establecido.

```
    def update(self):
        try:
            self.cur.execute("UPDATE notas SET notas=6.4 WHERE\
            idnotas=3")
            self.cur.execute("UPDATE notas SET notas=5.2 WHERE\
            idnotas=8")

        except psy.Error as e:
            print('Error Actualizar dato: %s', str(e))

        self.conn.commit()
```

Una vez ejecutada se obtiene lo siguiente:

Mostrar: notas

Visualizar contenido | Mostrar estructura | Modificar tabla | Nuevo Registro

Mostrar | Condición | Ordenar | Limite 50 | Longitud de texto 100 | Acción: Mostrar

`SELECT * FROM "notas" LIMIT 50` (0.000 s) Modificar

Modify	idnotas	name	edad	notas	idedic
modificar	1	Isabel Maniega	30	5.6	1
modificar	2	José Manuel Peña	30	7.8	1
modificar	4	Julia García	22	7.3	1
modificar	5	Amparo Mayora	28	8.4	3
modificar	6	Juan Martínez	30	6.8	3
modificar	7	Fernando López	35	6.1	2
modificar	3	Pedro López	25	6.4	2
modificar	8	María Castro	41	5.2	3

print_table(self, table):

En este caso se le pide al usuario que establezca que tabla quiere visualizar. La función imprimirá la información en esta tabla con un formato sencillo tal y como se muestra a continuación.

```python
    def print_table(self, table):
        try:
            self.cur.execute(f"SELECT * FROM {table};")
            rows = self.cur.fetchall()
            print(f'**{table}**')
            for row in rows:
                print(f'\t{row}')

        except psy.Error as e:
            print('Error Actualizar dato: %s', str(e))
```

```
**edicion**
        (1, 'Uno')
        (2, 'Dos')
        (3, 'Tres')
**notas**
        (1, 'Isabel Maniega', 30, 5.6, 1)
        (2, 'José Manuel Peña', 30, 7.8, 1)
        (4, 'Julia García', 22, 7.3, 1)
        (5, 'Amparo Mayora', 28, 8.4, 3)
        (6, 'Juan Martínez', 30, 6.8, 3)
        (7, 'Fernando López', 35, 6.1, 2)
        (3, 'Pedro López', 25, 6.4, 2)
        (8, 'María Castro', 41, 5.2, 3)
```

find_range(self, min, max, column, table):

Esta función requiere que el usuario introduzca un valor mínimo, uno máximo, la columna en la que buscar y la tabla a la que pertenece la misma, tras ello mostrará por pantalla resultados de la búsqueda con un formato sencillo.

```python
    def find_range(self, min, max, column, table):
        self.cur.execute(f"SELECT * FROM {table} WHERE\
         {column} >= {min} AND {column} <= {max}")
        print(
            f"Los registros de la tabla {table} cuyos valores en la columna\n\
            {column} estan entre {min} y {max} son: ")
        rows = self.cur.fetchall()
        for row in rows:
            print(f'\t{row}')
```

Obteniendo el siguiente resultado en comparación con el obtenido mediante Adminer para las notas entre el 5 y el 6,5:

```
Los registros de la tabla notas cuyos valores en la columna notas estan entre 5 y 6.5 son:
        (1, 'Isabel Maniega', 30, 5.6, 1)
        (7, 'Fernando López', 35, 6.1, 2)
        (3, 'Pedro López', 25, 6.4, 2)
        (8, 'María Castro', 41, 5.2, 3)
```

O el siguiente resultado en comparación con el obtenido mediante Adminer para los alumnos pertenecientes a la segunda edición:

```
Los registros de la tabla notas cuyos valores en la columna idedic estan entre 2 y 2 son:
        (7, 'Fernando López', 35, 6.1, 2)
        (3, 'Pedro López', 25, 6.4, 2)
```

delete(self):

Elimina el registro con el dato seleccionado.

```
    def delete(self):
        try:
            self.cur.execute("DELETE FROM notas WHERE\
            name='Pedro López';")
        except psy.Error as e:
            print('Error eliminar dato: %s', str(e))

        self.conn.commit()
```

close_conection(self):

Cierra la conexión con el servidor una vez hemos terminado de trabajar con el.

```
    def close_conection(self):
        self.cur.close()
        self.conn.close()
```

Información extra

Al igual que hice con el módulo de SQLite3, aprovecharé el código creado en esta lección para generalizarlo y crear un módulo que simplifique el uso de psycopg2 ya que tener que crear todas esas lineas de código para realizar tareas sencillas con las bases de datos me da la sensación de que se queda un poco fuer a de la filosofía de Python en la que uno de los puntos es la sencillez del código.